$P \stackrel{?}{=} NP$

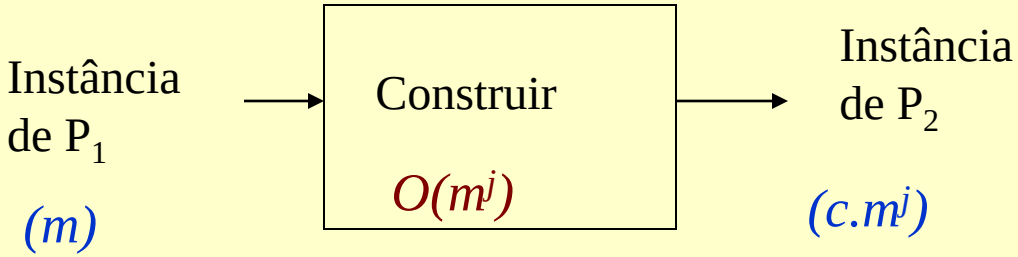
- Uma das principais questões em aberto é se $P = NP$, isto é, se de fato tudo o que pode ser feito em tempo polinomial por uma MTND poderia ser feito por uma MTD em tempo polinomial, talvez com um polinômio de grau mais alto.

Como saber se um problema está em NP e não está em P?

- Se sabemos que um problema $\mathbf{P_1 \in NP}$, e queremos saber se $P_2 \in P$ ou $P_2 \in NP$, então:
  - Se for possível **reduzir** P_1 a P_2 em tempo polinomial, então $\mathbf{P_2 \in NP}$.

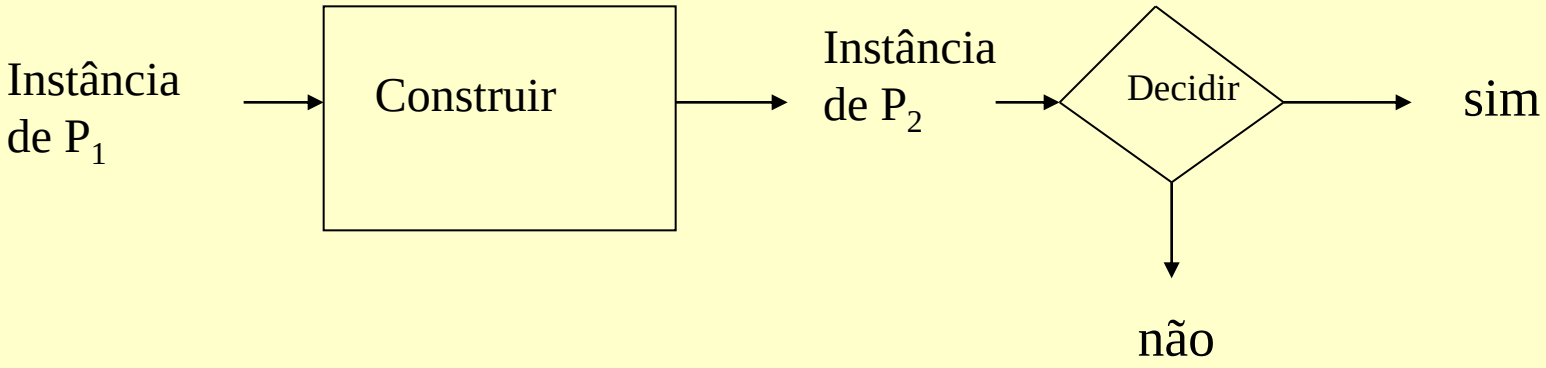
Redução de tempo polinomial

- Uma redução de P_1 a P_2 é em tempo polinomial se ela leva um tempo igual a algum polinômio no comprimento da instância de P_1.
- **Consequência**: a instância de P_2 terá um comprimento polinomial no comprimento da instância de P_1.



Por que a redução tem que ser em tempo polinomial para P_2 herdar a propriedade de P_1?

- Porque, dependendo da complexidade da redução, a herança pode não ocorrer.

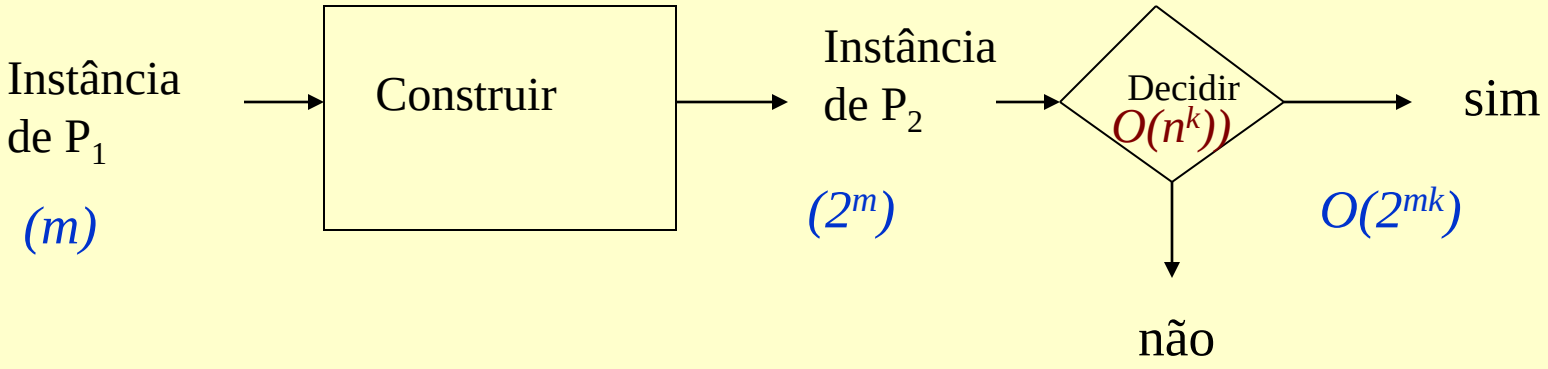


Gostaríamos de mostrar que $P_2 \in$ NP se $P_1 \in$ NP. Com a redução, se o algoritmo de decisão de P_2 for polinomial, então $P_2 \in P$ e concluiríamos que $P_1 \in$ P, o que seria um absurdo.

No entanto, só a existência do algoritmo de redução não é suficiente....

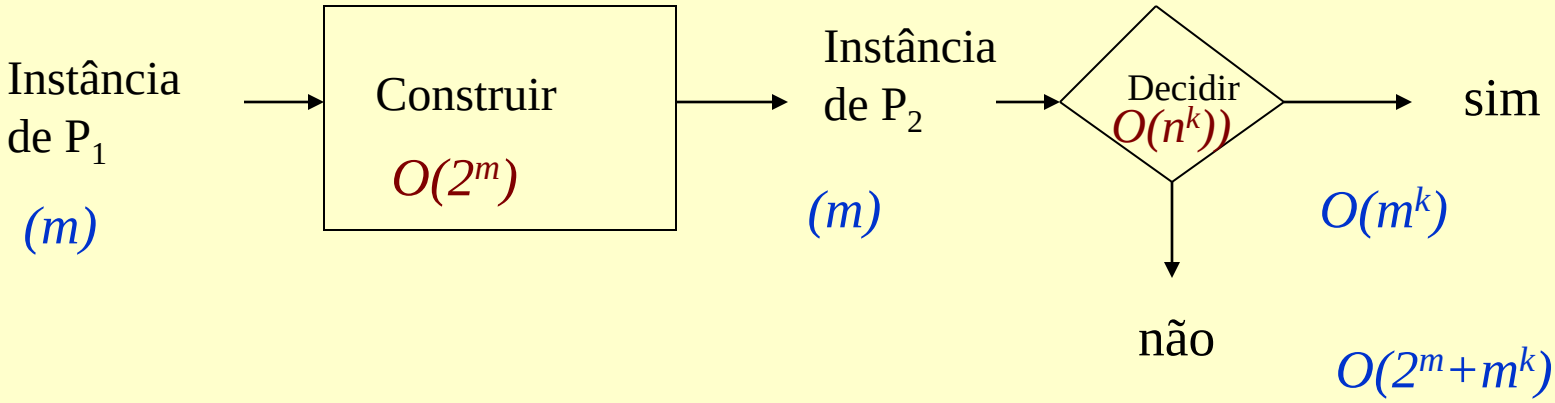
Se "Decidir" é polinomial *(O(n^k))* e:

- "Construir" produz, para uma instância de P_1 de comprimento *m*, uma saída de comprimento 2^m. Então "Decidir" vai ser exponencial *O(2^{mk}).* Assim, o algoritmo de decisão para P_1 demora, quando recebe uma entrada de comprimento *m*, um tempo exponencial em *m*. Isso é consistente com a situação em que P_2 está em *P* e P_1 não está em *P*. Logo, não há herança.



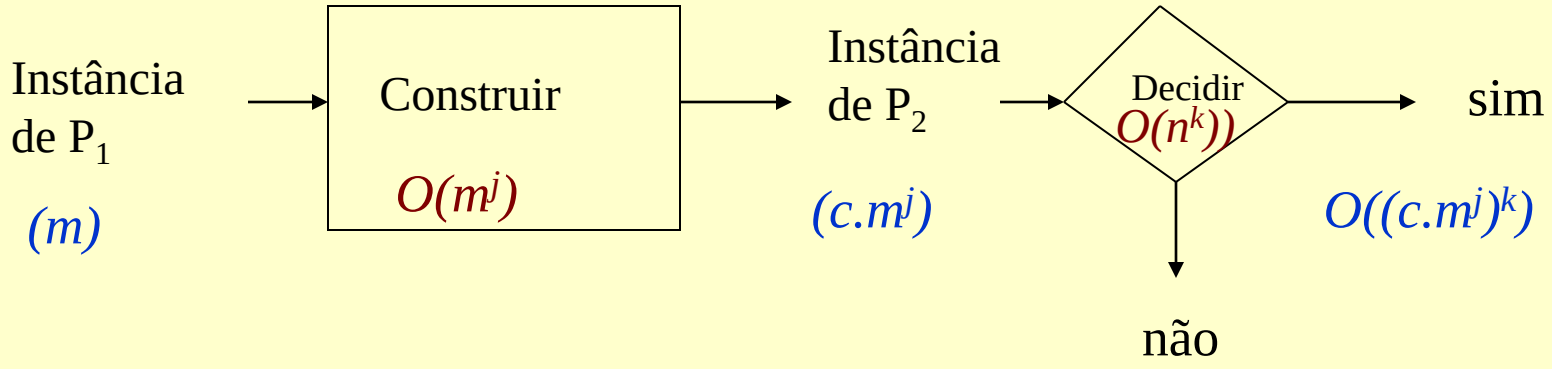
Se “Decidir” é polinomial *($O(n^k)$)* e:

- “Construir” produz, para uma instância de P_1 de comprimento *m*, uma saída também de comprimento *m*, mas demore para isso um tempo exponencial, digamos *$O(2^m)$*. Então mesmo se “Decidir” for polinomial em *m*, a única implicação é que um algoritmo de decisão para P_1 vai demorar ***$O(2^m + m^k)$*** sobre a entrada de comprimento *m*. Isso é novamente consistente com a situação em que P_2 está em *P* e P_1 não está. Logo, não há herança.



Conclusão

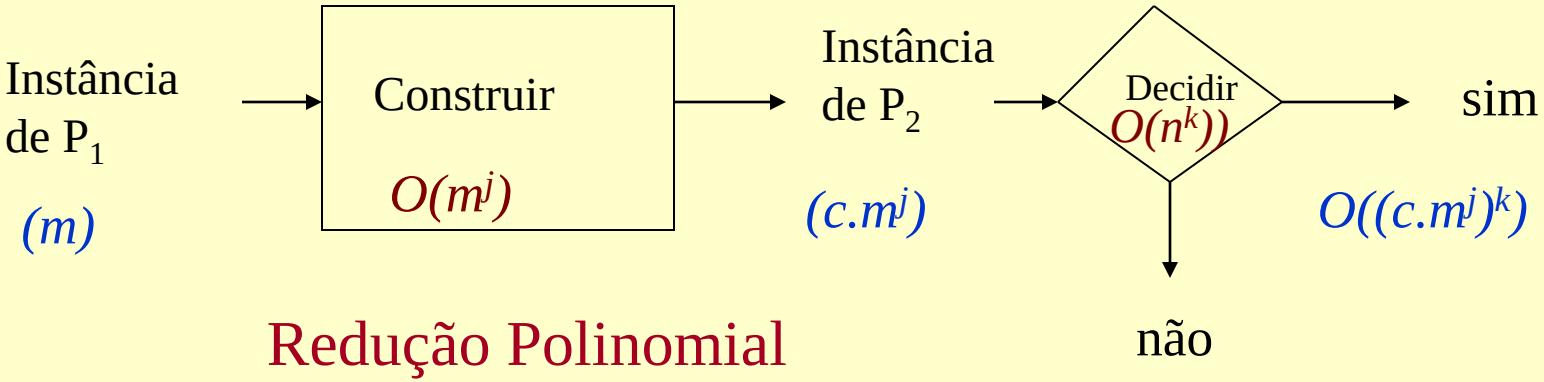
- A simples existência do algoritmo de conversão não nos garante que P_2 herda a propriedade de intratabilidade de P_1.
- Na teoria da intratabilidade, a redução só permite a herança das propriedades de P_1 para P_2 se o tempo de conversão ("Construir") das instâncias de P_1 para P_2 for polinomial no comprimento de sua entrada.
- Note que, se a conversão leva o tempo $O(m^j)$ sobre a entrada de comprimento m, então a instância de saída de P_2 não pode ser mais longa que o número de etapas tomadas, ou seja, ela é no máximo $c.m^j$ para alguma constante c.



Agora podemos provar:

"se P_2 está em *P*, então P_1 também está". Como P_1 *não* está em *P*, poderíamos então afirmar que P_2 também não está em *P*.

- Suponha que "Decidir" leva $O(n^k)$ para decidir sobre uma entrada de comprimento *n*. Então, podemos resolver a pertinência a P_1 de uma cadeia de comprimento *m* no tempo $O(m^j + (c.m^j)^k)$, onde m^j define o tempo para realizar a conversão, e o termo $(c.m^j)^k$ é o tempo para resolver a instância resultante de P_2.
- $O(m^j+cm^{jk})$, com *c*, *j*, *k* constantes, é um polinômio em *m*, e concluímos que P_1 está em *P* (Absurdo!). Logo, P_2 está em *NP*.



Redução Polinomial

Problemas NP-completos

- P é a subclasse de "menor dificuldade" de NP. Uma subclasse de "maior dificuldade" de NP é a classe dos problemas NP-completos. Intuitivamente, se uma solução polinomial for encontrada para um problema NP-completo, então todo problema de NP também admite solução polinomial (e consequentemente, nessa hipótese, P=NP).

- Seja L uma linguagem (um problema) em NP. Dizemos que **L é *NP-completa*** se:
  1. L está em NP.
  2. Para toda linguagem L' em NP, existe uma redução de tempo polinomial de L' a L.

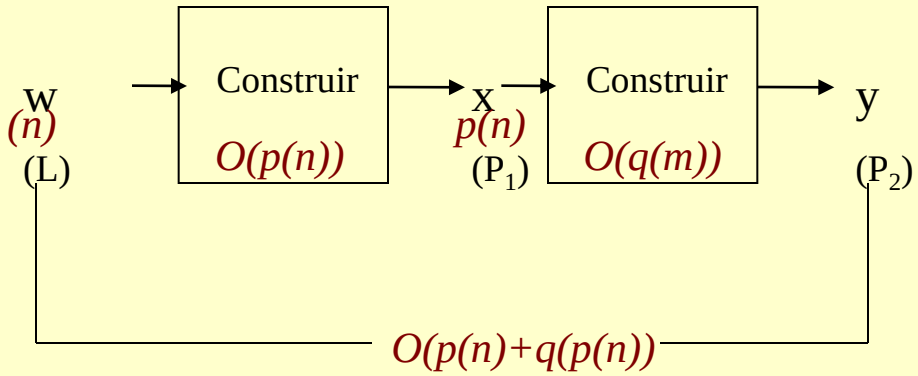
- **Teorema 1**: Se P_1 é *NP-completo*, P_2 está em *NP* e existe uma redução de tempo polinomial de P_1 a P_2, então P_2 é *NP-completo.*
- Prova: precisamos mostrar que toda linguagem L de *NP* se reduz em tempo polinomial a P_2 *(pela definição de NP-completo)*

Continuação da prova....

Como P_1 é *NP-completo*, sabemos que existe uma redução de tempo polinomial (*p(n)*) de L (qualquer) a P_1. Assim, uma cadeia w em L de comprimento *n* é convertida numa cadeia x em P_1 de comprimento no máximo igual a *p(n)*.

Por hipótese, existe uma redução de tempo polinomial *(q(m))* de P_1 a P_2. Então essa redução transforma x em alguma cadeia y de P_2 em no máximo tempo *q(p(n))*. Assim, a transformação de w em y demora um tempo no máximo igual a *p(n)+q(p(n))*, que é um polinômio em *n*.

Concluímos que L é redutível em tempo polinomial a P_2. Como L pode ser qualquer linguagem de *NP*, mostramos que tudo o que está em *NP* se reduz em tempo polinomial a P_2; isto é, P_2 é *NP-completo*.



- **Teorema 2**: Se algum problema *NP-completo* P estiver em *P* então *P=NP*.
- *Ou seja, se qualquer problema de NP estiver em P, então todos os problemas de NP também estarão em P.*
- Prova: Suponha que P seja *NP-completo* e ao mesmo tempo esteja em *P*. Então todas as linguagens L em *NP* se reduzem em tempo polinomial a P. Se P está em *P*, então L também estaria em *P*.
- Como se acredita que $P \neq NP$, ou seja, que existem muitos problemas em *NP* que não estão *P*, consideramos uma prova de que um problema é *NP-completo* equivalente a uma prova de ele não ter nenhum algoritmo polinomial (i.e. não pertence a *P*) e, portanto, nenhuma boa solução por computador.
- Os problemas NP-completos são vistos como uma generalização de todos os problemas de NP.

Problemas NP-difíceis (NP-hard)

- Alguns problemas L são tão difíceis que, embora possamos provar a condição (2) da definição de *NP-completo* (toda linguagem em *NP* se reduz a L em tempo polinomial), não podemos provar a condição (1): que L está em *NP (existe uma MTND)*. Nesse caso, dizemos que L é *NP-difícil (NP-hard)* (pode-se usar o termo "intratável" no sentido de *NP-difícil*).

Efeitos da NP-completude

1. Quando descobrimos que um problema é *NP-completo*, ele nos diz que existe pouca chance de um algoritmo eficiente poder ser desenvolvido para resolvê-lo. Somos encorajados a procurar por heurísticas, soluções parciais, aproximações ou outros meios. Além disso, podemos fazer isso com a confiança de que não estamos apenas "trapaceando".
2. Cada vez que adicionamos um novo problema *NP-completo*, P, à lista, reforçamos a ideia de que *todos* os problemas *NP-completos* exigem tempo exponencial num computador. O esforço que foi despendido na busca de um algoritmo de tempo polinomial para o problema P foi, não intencionalmente, um esforço dedicado a mostrar que $P=NP$. O resultado desse esforço é a grande evidência de que a) é muito improvável que $P=NP$, e b) *todos* os problemas *NP-completos* exigem tempo exponencial.

Problemas sobre Grafos *NP-completos*

- O Problema do Caixeiro Viajante (encontrar um Ciclo Hamiltoniano)
- O Problema da Cobertura de Nós: *encontrar um conjunto de nós tal que cada aresta do grafo tem pelo menos uma de suas extremidades em um nó do conjunto.*
- O Problema CLIQUE: *verificar se um grafo tem um k-clique, ou seja, um conjunto de k nós tal que existe uma aresta entre todo par de nós no clique.*
- O Problema da Coloração: *um grafo G pode ser "colorido" com k cores?*

- O Problema da Mochila: *dada uma lista de k inteiros, podemos particioná-los em dois conjuntos cujas somas sejam iguais?*
- O Problema do Escalonamento do tempo de execução unitário: *dadas k tarefas T_1, T_2, ...T_k, uma série de "processadores" p, um tempo limite t, e algumas restrições de precedência, da forma $T_i<T_j$ entre tarefas, existe um escalonamento de tarefas tal que: 1) cada tarefa seja atribuída a uma unidade de tempo entre 1 e t; 2) no máximo p tarefas sejam atribuídas a qualquer unidade de tempo, e 3) as restrições de precedência sejam respeitadas: se $T_i<T_j$, então T_i é atribuída a uma unidade de tempo anterior a T_j ?*

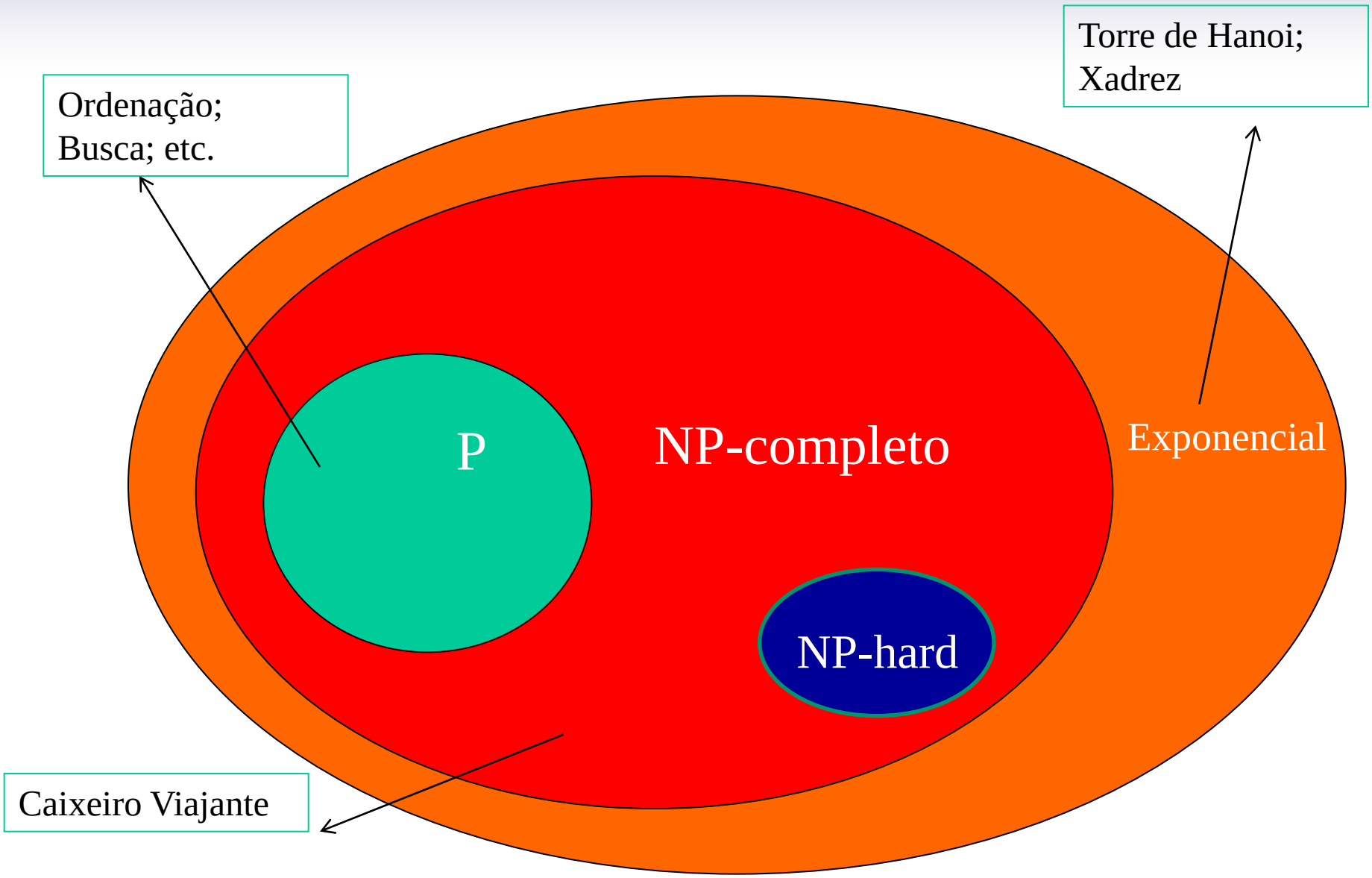
Resumo

- *As classes P e NP*. *P* consiste de todas as linguagens ou problemas aceitos por alguma MT que funciona em algum período de tempo polinomial, como uma função do comprimento de sua entrada. *NP* é a classe de linguagens ou problemas que são aceitos por MTND com um limite polinomial sobre o tempo que leva qualquer sequência de escolhas não-determinísticas.
- *A questão P=NP*. Não se sabe se *P* e *NP* são realmente as mesmas classes de linguagens, embora existam fortes suspeitas de que existem linguagens em *NP* que não estão em *P*.
- *Reduções de tempo polinomial.* Se podemos transformar instâncias de um problema em tempo polinomial em instâncias de um segundo problema que tem a mesma resposta – sim ou não – dizemos que o primeiro problema é redutível em tempo polinomial ao segundo.

Resumo

- *Problemas NP-completos:* Uma linguagem é *NP-completa* se está em *NP* e se existe uma redução de tempo polinomial de cada linguagem em *NP* à linguagem em questão. O fato de ninguém jamais ter encontrado um algoritmo de tempo polinomial para qualquer um dos milhares de problemas *NP-completos* conhecidos dá força à crença de que nenhum problema *NP-completo* está em *P*.
- *Problemas de satisfatibilidade NP-completos:* O Teorema de Cook mostrou o primeiro problema NP-completo – se uma expressão booleana é satisfatível – pela redução de todos os problemas em NP ao problema SAT em tempo polinomial.
- *Outros Problemas NP-completos:* Existe uma vasta coleção de problemas NP-completos conhecidos (caixeiro viajante, circuito hamiltoniano, cobertura de nós, etc.); provou-se que cada um deles é NP-completo por meio de uma redução de tempo polinomial de algum problema NP-completo conhecido.

Complexidade de Algoritmos



Complexidade de Algoritmos

Exemplo:

400 estudantes universitários pleiteiam acomodação no alojamento, que acomoda apenas 100. Para complicar, o reitor forneceu uma lista com pares de estudantes que não podem figurar na lista dos escolhidos. Deseja-se uma lista-solução com 100 estudantes que podem ocupar o alojamento.

Este é um problema NP, uma vez que encontrar uma solução a partir da lista dos 400 candidatos e da lista de pares incompatíveis do reitor consiste num algoritmo exponencial. No entanto, o problema de decisão correspondente (dada uma lista de 100 estudantes, certificar se ela é uma solução possível) tem certificação polinomial.

Complexidade de Algoritmos

Vejamos:

- Para certificar, basta verificar que nenhum par de estudantes da lista do reitor ocorre na lista candidata.
- Já para construir uma lista-solução, seria necessário certificar cada uma das diferentes combinações de 100 estudantes, a partir dos 400 estudantes. Esse número é maior que o número de átomos do universo! Assim, nenhum supercomputador do futuro será capaz de solucionar esse problema por força bruta.

Complexidade de Algoritmos

Na verdade:

Este problema é modelado como um grafo (cada vértice corresponde a um estudante; cada aresta liga dois estudantes que não podem ficar juntos no alojamento), e sua solução corresponde em verificar, para 100^{400} possibilidades, se é possível "colorir" o grafo com 100 cores.